CS522 - Programming Language Semantics

Cartesian Closed Categories as Models for Simply-Typed λ-Calculus

Grigore Roşu
Department of Computer Science
University of Illinois at Urbana-Champaign

We next focus on showing that CCCs can be organized as models of simply-typed λ-calculus. The idea is to interpret types as objects and λ-expressions as morphisms. As expected, exponentiation is going to play a crucial role. Indeed, if A and B are objects types interpreting s and t, respectively, then B^A will interpret the type $s \to t$.

To make our categorical framework and modeling more interesting, in the sequel we consider a common extended variant of simply typed λ-calculus, namely one with a *set* of type constants (as opposed to just one type constant, $\circ$, like before), a product type constructor, and a set of typed λ-constants.

Formally, let S be a set of *type constants*, also possibly called *basic* or *primitive types*. The extended λ-calculus that we consider from here on has types built over the following grammar:

$$Type \quad ::= \quad S \quad | \quad Type \to Type \quad | \quad Type \times Type$$

As expected, types $s \times t$ stand for pairs of values, one of type s and the other of type t.

Let us also consider a set Σ of *typed λ-constants*, that is, pairs $\sigma : t$, where t is some type. Like for type assignments, we assume that Σ does not contain constants with multiple types and let $\Sigma(\sigma)$ denote the type t such that $\sigma : t \in \Sigma$. We also take the liberty to write $\sigma \in \Sigma$, without mentioning the type of σ. Then the formal CFG of (possibly not well-typed) λ-expressions is:

$$\begin{aligned} Exp \quad ::= \quad & Var \mid \Sigma \mid \lambda\, x\!:\!s.Exp \mid ExpExp \mid \\ & \mid (Exp, Exp) \mid proj_1 Exp \mid proj_2 Exp \end{aligned}$$

The set of constant λ-expressions Σ is also called a *signature*. Besides the constants in the signature, our extended simply-typed λ-calculus also includes the pairing construct, as well as the two corresponding projections. The typing rules for the new constructs are as follows:

$$X \rhd \sigma : s$$ for any X and $\sigma : s \in \Sigma$;

$$\frac{X \rhd E_1 : t_1 \qquad X \rhd E_2 : t_2}{X \rhd (E_1, E_2) : t_1 \times t_2}$$ for any X, any Exp-terms E_1, E_2, and any types t_1, t_2;

$$\frac{X \rhd E : t_1 \times t_2}{X \rhd proj_1 E : t_1}$$ for any type assignment X, Exp-term E, and types t_1, t_2;

$$\frac{X \rhd E : t_1 \times t_2}{X \rhd proj_2 E : t_2}$$ for any type assignment X, Exp-term E, and types t_1, t_2.

Two equations need to be added, to capture the relationship between pairing and projections:

$(\mathsf{Proj})_1$ $\quad \mathcal{E} \vdash (\forall X)\ proj_1(E_1, E_2) =_{t_1} E_1$ if $X \triangleright E_1 : t_1$ and $X \triangleright E_2 : t_2$;

$(\mathsf{Proj})_2$ $\quad \mathcal{E} \vdash (\forall X)\ proj_2(E_1, E_2) =_{t_2} E_2$ if $X \triangleright E_1 : t_1$ and $X \triangleright E_2 : t_2$.

Exercise 1 *Show that it is not necessarily the case that* $\mathcal{E} \vdash (\forall X)(proj_1 E, proj_2 E) =_{t_1 \times t_2} E$ *whenever* $X \triangleright E : t_1 \times t_2$. *What if the types* s *associated to constants* $\sigma : s$ *in* Σ *are such that they contain no product type construct?*

Simply-Typed λ-Calculus Captures Algebraic Specifications

The extension of simply-typed λ-calculus defined above captures algebraic signatures quite elegantly. Given a many-sorted algebraic signature (S, Σ), i.e., a set of sorts S and a set of operations $\sigma : s_1 \times \ldots s_n \rightarrow s$ (we define algebraic signatures in Maude, for example), all one needs to do is to declare the sorts in S as basic types and the operations in Σ as λ-constants, noting that $s_1 \times \ldots s_n$ are types that can be constructed in λ-calculus (assume right or left associativity for the product of types).

One can show that the simply-typed λ-calculus obtained this way is a *conservative extension* of the original equational theory, in the sense that it can derive precisely all the equational consequences of the equational theory.

(S, Σ)-CCC

The simply-typed λ-calculus defined above is therefore parameterized by the pair (S, Σ) of basic types and constant λ-expressions. We next show that any CCC providing interpretations for just the basic types in S and for the λ-constants in Σ can be extended to a model of (simply-typed) λ-calculus, providing interpretations for all the types and for all the λ-expressions. This is possible because the CCCs provide elegant corresponding categorical machinery to the syntactic constructs of types and λ-expressions.

An (S, Σ)-*CCC* is a CCC $\mathcal{C}$ together with

- a mapping $[\![\_]\!] : S \to |\mathcal{C}|$, associating some object $[\![s]\!] \in |\mathcal{C}|$ to any $s \in S$;

- some morphism $[\![\sigma]\!] : \star \to [\![t]\!]$ for any $\sigma : t \in \Sigma$, where $[\![\_]\!] : \mathit{Type} \to |\mathcal{C}|$ extends the map above as follows:
  - $[\![s \to t]\!] \stackrel{def}{=} [\![t]\!]^{[\![s]\!]}$, and
  - $[\![s \times t]\!] \stackrel{def}{=} [\![s]\!] \times [\![t]\!]$.

Recall that $\star$ is the final object of $\mathcal{C}$.

We show that the operator $[\![\_]\!] : \Sigma \to \mathcal{C}$ above can be extended to any λ-expressions. Our goal is therefore to define for each well-typed λ-expression $X \triangleright E : t$, where $X = x_1 : s_1, \ldots, x_n : s_n$, a morphism $[\![X \triangleright E : t]\!] : [\![s_1]\!] \times \ldots [\![s_n]\!] \to [\![t]\!]$ in $\mathcal{C}$.

Before that, let us first discuss how such morphisms would relate to the more natural interpretation of well-typed λ-expressions in Henkin models, namely as functions $M_{X \triangleright E : t} : [X \to M] \to M_t$ on *environments*. The key observation here is the set of M-environments $[X \to M]$ is in a one-to-one correspondence with

the set $M_{s_1} \times \ldots \times M_{s_n}$. Indeed, an M-environment is nothing but a choice of an element in each of the sets $M_{s_1}, \ldots, M_{s_n}$, which is nothing but an element in $M_{s_1} \times \ldots \times M_{s_n}$. Therefore, we regard the functions $M_{X \triangleright Et}$ that appear in a Henkin model as functions $M_{s_1} \times \ldots \times M_{s_n} \to M_t$, which now look closely related to our morphisms $[\![X \triangleright E{:}t]\!] : [\![s_1]\!] \times \ldots \times [\![s_n]\!] \to [\![t]\!]$ defined next. We will actually see later on that there is a very tight relationship between the CCC models defined next and Henkin models.

Recall that $A \times B$ and $B \times A$ are isomorphic in $\mathcal{C}$. Therefore, the objects $[\![s_1]\!], \ldots, [\![s_n]\!]$ can be permuted in any order in the product $[\![s_1]\!], \ldots, [\![s_n]\!]$. However, in what follows, we prefer to work with type assignments as lists rather than sets; this way we can generate fixed interpretations rather than the possibly confusing "up-to-isomorphism" ones. To accomplish this, we need to slightly modify the inference rules to account for lists rather than sets.

Fortunately, this is quite easy. We only need to replace each "$X, x:s$" by "$X_1, x:s, X_2$" correspondingly in the premise of rules, where X_1 and X_2 are any type assignments. For example, the rule (ξ) becomes

$$\frac{\mathcal{E} \vdash (\forall X_1, x:s, X_2)\ E =_t E'}{\mathcal{E} \vdash (\forall X_1, X_2)\ \lambda\, x:s.E =_{s\to t} \lambda\, x:s.E'}$$

Now we can extend $[\![\_]\!] : \mathit{Type} \to |\mathcal{C}|$ to type assignments, i.e., $[\![\_]\!] : \mathit{TypeAssignment} \to |\mathcal{C}|$, as follows:

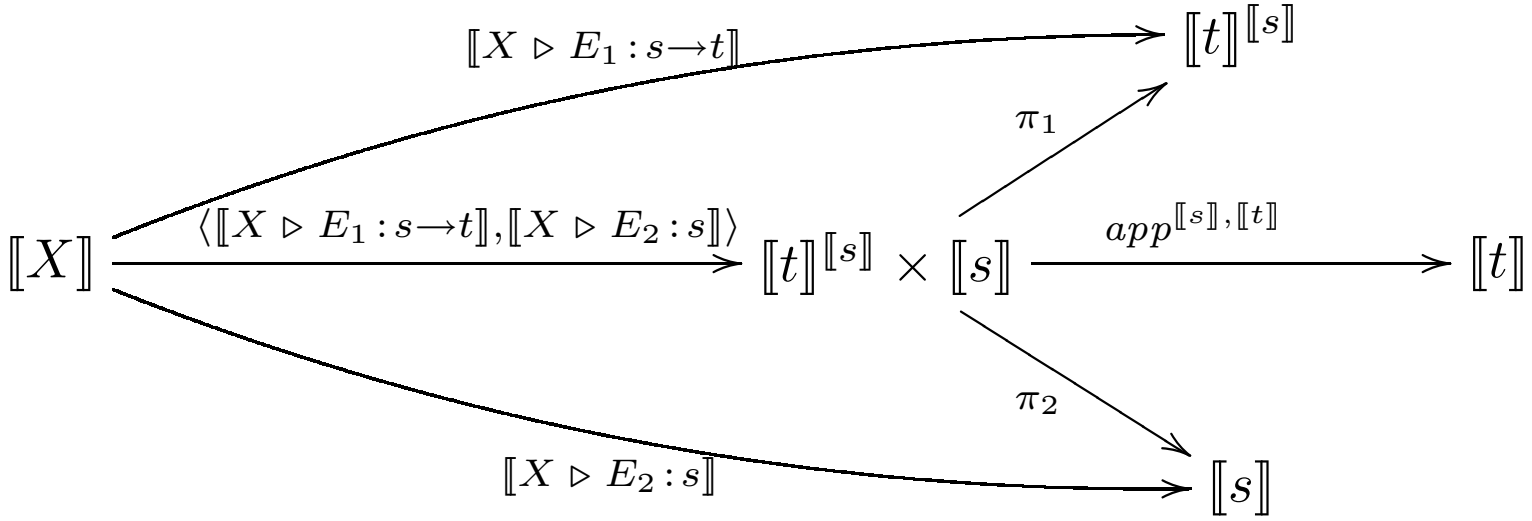
$$[\![x_1:s_1, \ldots, x_n:s_n]\!] \stackrel{def}{=} [\![s_1]\!] \times \ldots \times [\![s_n]\!]\ .$$

We are now ready to define the morphisms $[\![X \ \triangleright\ E:t]\!] : [\![X]\!] \to [\![t]\!]$ in $\mathcal{C}$, inductively on the structure of well-typed λ-expressions:

- $[\![X_1, x_i:s_i, X_2 \ \triangleright\ x_i:s_i]\!] \stackrel{def}{=} \pi_i$, the i-th projection given by the definition of the product, i.e., $\pi_i : [\![X_1]\!] \times [\![s_i]\!] \times [\![X_2]\!] \to [\![s_i]\!]$;

- $[\![X \triangleright \sigma : t]\!] \stackrel{def}{=} !_{[\![X]\!]}; [\![\sigma]\!]$ for any $\sigma : t \in \Sigma$, where $!_{[\![X]\!]} : [\![X]\!] \to \star$ is the unique morphism from $[\![X]\!]$ to the final object $\star$, and $[\![\sigma]\!] : \star \to [\![t]\!]$ is the interpretation of σ given as part of the definition of the (S, Σ)-CCC $\mathcal{C}$. Intuitively, the morphism $[\![X \triangleright \sigma : t]\!]$ "forgets" its input and "recalls" the "built in" σ;
- $[\![X \triangleright \lambda x : s.E : s \to t]\!] \stackrel{def}{=} curry([\![X, x : s \triangleright E : t]\!])$. Note that, indeed, this morphism is well-defined: this is because $[\![X, x : s \triangleright E : t]\!]$ is a morphism $[\![X]\!] \times [\![s]\!] \to [\![t]\!]$ and *curry* takes morphisms in $\mathcal{C}([\![X]\!] \times [\![s]\!], [\![t]\!])$ to morphisms in $\mathcal{C}([\![X]\!], [\![t]\!]^{[\![s]\!]})$. Therefore, $curry([\![X, x : s \triangleright E : t]\!])$ is a morphism $[\![X]\!] \to [\![s \to t]\!]$, as expected. Note the elegance of the definition of this CCC interpretation, in contrast to the corresponding definition in Henkin models;
- $[\![X \triangleright E_1 E_2 : t]\!] \stackrel{def}{=} \langle [\![X \triangleright E_1 : s \to t]\!], [\![X \triangleright E_2 : s]\!] \rangle; app^{[\![s]\!],[\![t]\!]}$. This definition needs some explanation. Note that

$[\![X \triangleright E_1 : s \to t]\!]$ is a morphism $[\![X]\!] \to [\![t]\!]^{[\![s]\!]}$, while
$[\![X \triangleright E_2 : s]\!]$ is a morphism $[\![X]\!] \to [\![s]\!]$, so
$\langle [\![X \triangleright E_1 : s \to t]\!], [\![X \triangleright E_2 : s]\!] \rangle$ is a morphism
$[\![X]\!] \to [\![t]\!]^{[\![s]\!]} \times [\![s]\!]$. This morphism can be indeed composed with $app^{[\![s]\!],[\![t]\!]} : [\![t]\!]^{[\![s]\!]} \times [\![s]\!] \to [\![t]\!]$, thus giving a morphism $[\![X]\!] \to [\![t]\!]$, as desired. All these morphisms are depicted in the following figure:



Like in the diagram above, it is often the case that constructions

and proofs in category theory are driven almost automatically by diagrams; note that there are no other simpler ways to put together the morphisms $[\![X]\!] \to [\![t]\!]^{[\![s]\!]}$ and $[\![X]\!] \to [\![s]\!]$ into a morphism $[\![X]\!] \to [\![t]\!]$.

The remaining definitions are straightforward:

- $[\![X \triangleright (E_1, E_2) : t_1 \times t_2]\!] \stackrel{def}{=} \langle [\![X \triangleright E_1 : t_1]\!], [\![X \triangleright E_2 : t_2]\!] \rangle$;
- $[\![X \triangleright proj_1 E : t_1]\!] \stackrel{def}{=} [\![X \triangleright E : t_1 \times t_2]\!]; \pi_1$;
- $[\![X \triangleright proj_2 E : t_2]\!] \stackrel{def}{=} [\![X \triangleright E : t_1 \times t_2]\!]; \pi_2$.

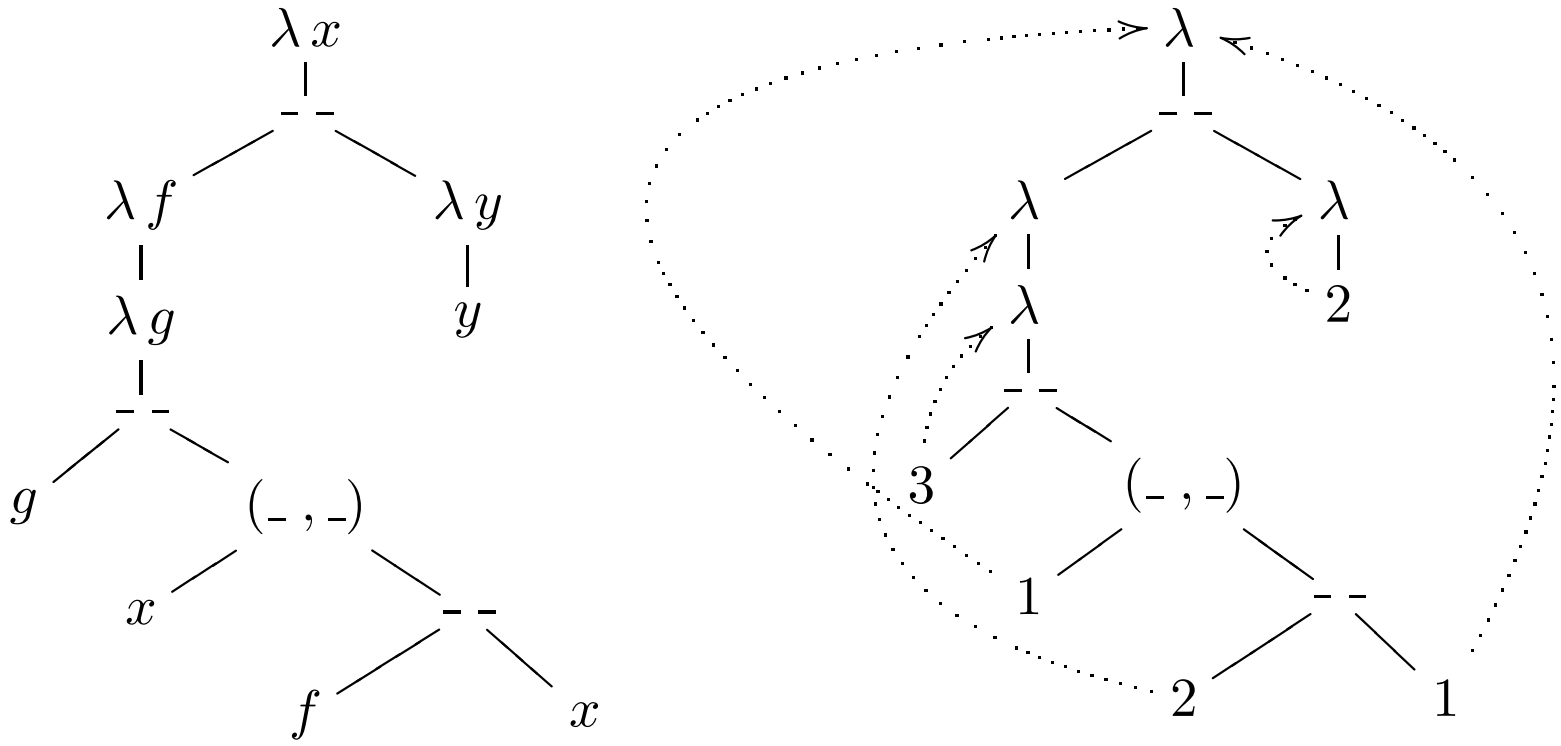
We can associate a morphism $[\![X \triangleright E : t]\!] : [\![X]\!] \to [\![t]\!]$ to any well-typed λ-expression $X \triangleright E : t$. Let us construct such a morphism for a concrete λ-expression:

$$
\begin{aligned}
&[\![\emptyset \triangleright \lambda x:s.(\lambda f:s \to s.\lambda g:s \times s \to t.g(x, fx))(\lambda y:s.y) : s \to (s \times s \to t) \to t]\!] = \\
&curry([\![x:s \triangleright (\lambda f:s \to s.\lambda g:s \times s \to t.g(x, fx))(\lambda y:s.y) : (s \times s \to t) \to t]\!]) = \\
&curry(\langle [\![x:s \triangleright \lambda f:s \to s.\lambda g:s \times s \to t.g(x, fx) : u]\!], [\![x:s \triangleright \lambda y:s.y : v]\!] \rangle ; app^{[\![u]\!],[\![v]\!]}) = \\
&curry(\langle curry(curry([\![x:s, f:s \to s, g:s \times s \to t \triangleright g(x, fx) : t]\!])), curry([\![x:s, y:s \triangleright y:s]\!]) \rangle ; app^{[\![u]\!],[\![v]\!]}) = \\
&curry(\langle curry(curry(\langle [\![X \triangleright g:s \times s \to s]\!], [\![X \triangleright (x, fx) : s \times s]\!] \rangle ; app^{[\![s \times s]\!],[\![s]\!]})), curry(\pi_2) \rangle ; app^{[\![u]\!],[\![v]\!]}) = \\
&curry(\langle curry(curry(\langle \pi_3, \langle \pi_1, \langle \pi_2, \pi_1 \rangle ; app^{[\![s]\!],[\![s]\!]} \rangle \rangle ; app^{[\![s \times s]\!],[\![s]\!]})), curry(\pi_2) \rangle ; app^{[\![u]\!],[\![v]\!]}),
\end{aligned}
$$

where $u \stackrel{def}{=} (s \to s) \to (s \times s \to t) \to t$, $v \stackrel{def}{=} s \to s$, and $X \stackrel{def}{=} x:s, f:s \to s, g:s \times s \to s$.

Interestingly, note that the morphism obtained above contains no references to the variables that occur in the original λ-expression. It can be shown that the interpretation of λ-expressions into a CCC is *invariant to α-conversion*. To see that, let us draw the morphism above as a tree, where we write λ instead of *curry*, _ _

instead of $\langle \_ , \_ \rangle$; $app^{[\![s]\!],[\![t]\!]}$, $(\_ , \_)$ instead of the remaining $\langle \_ , \_ \rangle$ and i instead of π_i (and omit the types):



The (right) tree above suggests a representation of λ-expressions that is invariant to α-conversion: each binding variable is replaced by a natural number, representing the number of λs occurring on the path to it; that number then replaces consistently all the bound

occurrences of the variable. The corresponding lambda expression without variables obtained using this transformation is $\lambda\,(\lambda\,\lambda\,(3\ (1,(2\ 1)))\ \lambda\,2)$.

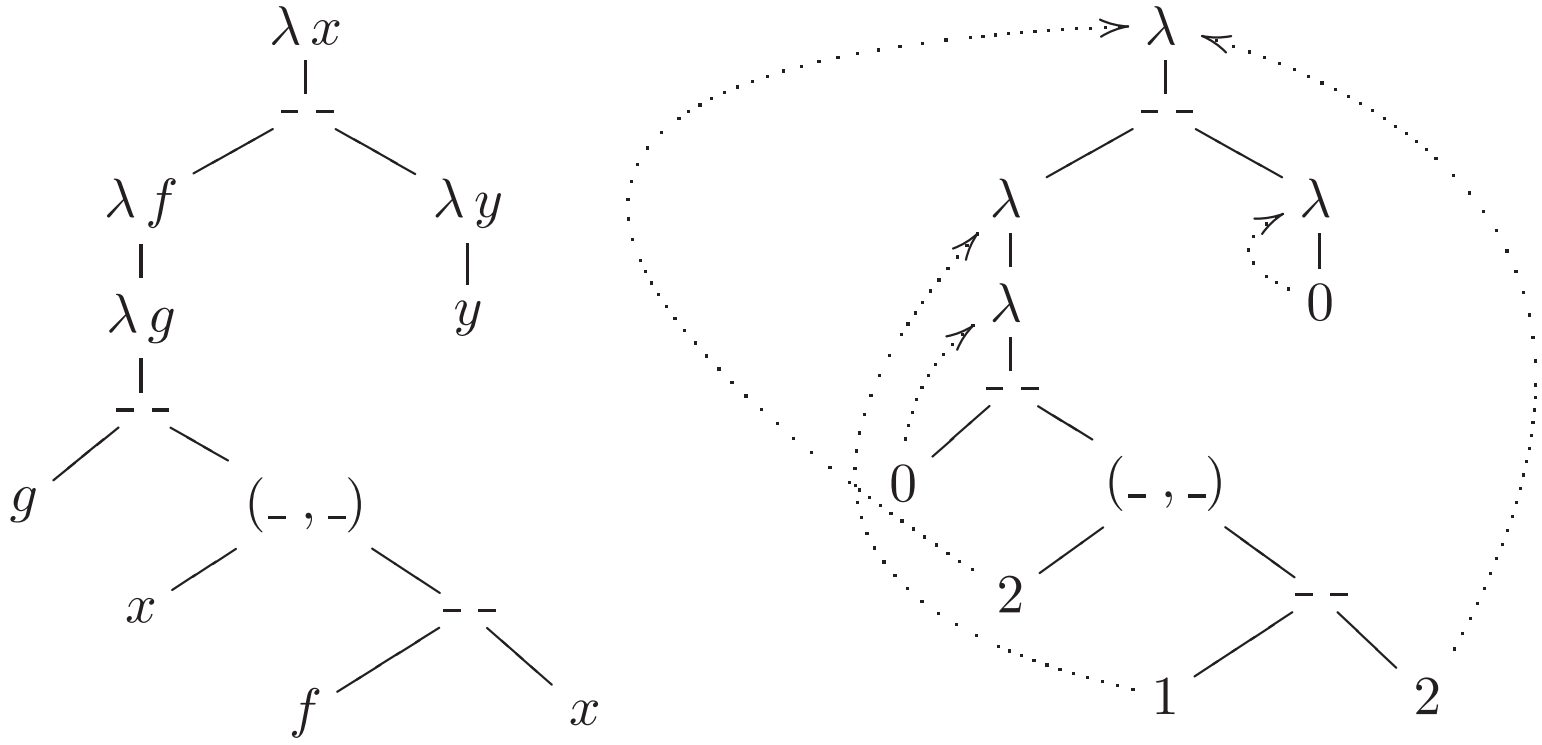
Exercise 2 *Explain why this representation is invariant to α-conversion.*

The representation of α-expressions above was explicitly proposed as a means to implement λ-calculus in 1971 by Nicholas de Bruijn.

In the same paper, de Bruijn proposed another encoding which became more popular. We do not know whether de Bruijn was influenced by the CCC interpretation of λ-expressions or not, but we discuss his other representation technique here.

de Bruijn Nameless Representation of λ-expression

The second and more popular representation technique of λ-expressions proposed by Nicholas de Bruijn in 1971 is a bottom-up version of the above representation. For the above example the tree representing the encoding is (we omit the types):



In this encoding, each variable is replaced by the number of lambda abstractions on the path from it to the lambda abstraction binding it. The encoding for the given example is $\lambda\,(\lambda\,\lambda\,(0\ (2,(1\ 2)))\ \lambda\,0)$. One can easily define application for the above de Bruijn encoding:

$(\lambda\,E\ E') \Rightarrow E[0 \leftarrow E']$
$(E\ E')[i \leftarrow E''] \Rightarrow (E[i \leftarrow E'])\ (E'[i \leftarrow E''])$ $(\lambda\,E)[i \leftarrow E'] \Rightarrow \lambda\,(E[(i+1) \leftarrow\uparrow E'])$ $j[i \leftarrow E] \Rightarrow$ if $j == i$ then E else (if $j > i$ then $j-1$ else j fi) fi
$\uparrow E' \Rightarrow\uparrow^0 E$
$\uparrow^i (E\ E') \Rightarrow (\uparrow^i E)\ (\uparrow^i E')$ $\uparrow^i (\lambda\,E) \Rightarrow \lambda\,(\uparrow^{i+1} E)$ $\uparrow^i j \Rightarrow$ if $j >= i$ then $(j+1)$ else j fi

Exercise 3 *Define the transformation above formally in Maude.*

Exercise 4 *Define application for the first de Bruijn encoding, in a similar style to the one above.*

Extending Henkin models

We can extend the definition of Henkin models to *(S, Σ)-Henkin* models. Thus, an *(S, Σ)-preframe* or *premodel (Henkin)* consists of the following:

- a set M_t for each type t;
- a (*Type*-indexed) mapping $M\_ : \Sigma \to M$;
- for any $s, t \in Type$
  - a function $M^{s,t} : M_{s \to t} \times M_s \to M_t$ which is *extensional*, i.e., for any $f, g \in M_{s \to t}$, if $M^{s,t}(f, u) = M^{s,t}(g, u)$ for all $u \in M_s$ then $f = g$;
  - two functions $M_s^{s \times t} : M_{s \times t} \to M_s$ and $M_t^{s \times t} : M_{s \times t} \to M_t$ which form an *extensional pair*, i.e., for any $p, q : M_{s \times t}$, if $M_s^{s \times t}(p) = M_s^{s \times t}(q)$ and $M_t^{s \times t}(p) = M_t^{s \times t}(q)$ then $p = q$;

Given $u \in M_s$ and $v \in M_t$, when it exists we let (u, v) denote the *unique* element in $M_{s\times t}$ such that $M_s^{s\times t}(p) = u$ and $M_t^{s\times t}(p) = v$.

A well-typed λ-expression $X \triangleright E : t$ is interpreted by a mapping $M_{X \triangleright E:t} : [X \to M] \to M_t$ such that:

- $M_{X \triangleright \sigma:s}(\rho) = M_\sigma \in M_s$;
- $M_{X \triangleright x:s}(\rho) = \rho(x : s) \in M_s$;
- $M^{s,t}(M_{X \triangleright \lambda\, x:s.E:s\to t}(\rho), v) = M_{X,x:s \triangleright E:t}(\rho[x \leftarrow v])$ for any $v \in M_s$;
- $M_{X \triangleright E_1 E_2:t}(\rho) = M^{s,t}(M_{X \triangleright E_1:s\to t}(\rho), M_{X \triangleright E_2:s}(\rho))$;
- $M_s^{s\times t}(M_{X \triangleright (E,E'):s\times t}(\rho)) = M_{X \triangleright E:s}(\rho)$ and $M_t^{s\times t}(M_{X \triangleright (E,E'):s\times t}(\rho)) = M_{X \triangleright E':t}(\rho)$;
- $M_{X \triangleright proj_1 E:s}(\rho) = M_s^{s\times t}(M_{X \triangleright E:s\times t}(\rho))$;
- $M_{X \triangleright proj_2 E:t}(\rho) = M_t^{s\times t}(M_{X \triangleright E:s\times t}(\rho))$.

Henkin models are CCCs

Given an (S, Σ)-Henkin model $\mathcal{M}$, one can define a (S, Σ)-CCC $\mathbb{C}$:

- the objects of $\mathbb{C}$: M_t for each $t \in \mathit{Type}$;
- the set of morphisms $\mathbb{C}(M_s, M_t)$ is $M_{s \to t}$;
- the identity morphism 1_{M_s} is $M_{\emptyset \,\triangleright\, \lambda\, x{:}s.x{:}s \to s}$;
- given $f \in \mathbb{C}(M_s, M_t)$ and $g \in \mathbb{C}(M_t, M_{t'})$, define $f; g$ as $M_{u:s \to t, v:t \to t' \,\triangleright\, \lambda\, x{:}s.v(ux){:}s \to t'}(\rho)$ where $\rho(u) = f$ and $\rho(v) = g$;
- $M_s \times M_t = M_{s \times t}$ and the projections are $M_s^{s \times t}$ and $M_t^{s \times t}$;
- the exponentiation object $M_t^{M_s}$ is $M_{s \to t}$, and the application morphism app^{M_s, M_t} is $M_{\emptyset \,\triangleright\, \lambda\, x{:}(s \to t) \times s.\mathit{proj}_1(x)\ \ \mathit{proj}_2(x){:}(s \to t) \times s \to t}$;
- $[\![t]\!] = M_t$ and $[\![\sigma]\!] = M_\sigma$.

Exercise 5 *Prove that $\mathbb{C}$ is indeed a (S, Σ)-CCC.*

Some CCCs are Henkin models

Let $\mathbb{C}$ be an (S, Σ)-CCC such that for each object A, $\mathbb{C}(\star, A)$ is a *homomorphic family*, that is, for any object B and any two morphisms $f, g : A \to B$, if $h; f = h; g$ for each $h : \star \to A$ then $f = g$; such a category is also called *well-pointed*. Then we can define a Henkin model $\mathcal{M}$ for (S, Σ) as follows:

- $M_t = \mathbb{C}(\star, [\![t]\!])$ for each type t;
- $M^{s,t} : M_{s \to t} \times M_s \to M_t$ is given by: for any $f : \star \to [\![s \to t]\!] = [\![t]\!]^{[\![s]\!]}$ and any $x : \star \to [\![s]\!]$, $M^{s,t}(f, x) = \langle f, x \rangle; app^{[\![s]\!],[\![t]\!]}$;
- given $X = x_1 : s_1, \ldots, x_n : s_n$ let $\prod_X = [\![s_1]\!] \times \cdots \times [\![s_n]\!]$. Each morphism $h : \star \to \prod_X$ is equivalent with the tuple $\langle h; \pi_{[\![s_1]\!]}, \ldots, h; \pi_{[\![s_n]\!]} \rangle$. Thus environments $\rho : X \to M$ and morphisms $\star \to \prod_X$ are in bijection. Let $\overline{\rho} : \star \to \prod_X$ denote

the image of ρ through this bijection;

- For each well-typed λ-expression $X \triangleright E : s$, and each assignment ρ, let $M_{X \triangleright E:s}(\rho) = \overline{\rho}; [\![X \triangleright E : s]\!]$.

Exercise 6 *Show that $\mathcal{M}$ is indeed an (S, Σ)-Henkin model.*